

Introduzione a Processing

di MATTEO FIORAVANTI

Conosciamo il linguaggio di programmazione open-source creato per realizzare immagini e animazioni, le cui librerie ed il cui ambiente sono compatibili con l'ambiente di sviluppo di Arduino. Prima puntata.

Processing è un linguaggio di programmazione sviluppato con il preciso scopo di rendere molto semplice l'interazione con elementi grafici; più esattamente, è finalizzato alla realizzazione di immagini, animazioni e interazioni. Ottenere questo con altri linguaggi, come ad esempio C++ o Java, risulta complesso e richiede un grado molto elevato di conoscenze. Gli ideatori e sviluppatori di Processing sono Casey Reas e Ben Fry: il primo è professore al dipartimento di Design And Media Arts alla UCLA, mentre Ben Fry è un designer e programmatore di Cambridge. Processing è stato sviluppato dal 2001, con la finalità di ottenere un linguaggio semplice per gli studenti di arte e design e per quelli ad indirizzo tecnico. Processing è stato notevolmente sviluppato

in questi anni, come pure le sue numerose librerie, che ad oggi coprono varie funzionalità: grafica 3D, comunicazione seriale, interfacciamento con Arduino, audio. Processing e le sue librerie sono completamente liberi, e come per gran parte di tutti i software open-source, la comunità di utilizzatori, appassionati, programmatori, hobbisti è molto ampia.
Generalmente quando si studia un linguaggio di programmazione si parte sostanzialmente dalla struttura, dai concetti teorici, dagli algoritmi e dai vari metodi di programmazione, relegando alla fine l'interazione con la grafica e magari le animazioni. Utilizzando Processing, si parte direttamente da grafica e animazioni, fermo restando che devono essere compresi le strutture di programmazione e i vari costrutti.
La piena compatibilità con Arduino è un altro aspetto che rende Processing estremamente interessante, senza contare che gli ambienti di programmazione sono praticamente identici. Con Arduino e Processing abbiamo la possibilità di realizzare delle applicazioni complete, in cui i dispositivi hardware vengono realizzati utilizzando Arduino e gli applicativi software sono scritti in Processing. Chi già conosce Arduino, troverà sicuramente familiare l'ambiente di Processing, dato che è praticamente lo stesso. Chi invece non conosce Arduino e volesse cominciare il suo percorso di apprendimento direttamente da Processing, non incontrerà alcuna difficoltà e potrà integrare successivamente i concetti di Arduino, se mai ne avesse necessità.

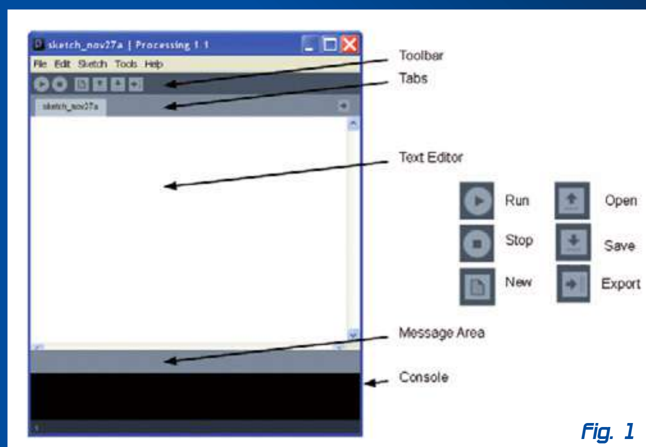


Fig. 1

Il corso che presentiamo ha lo scopo di fornire a studenti ed hobbisti le nozioni base per utilizzare Processing e per realizzare dei programmi che si interfaccino con Arduino. Alla fine di questo corso si avranno le nozioni base per realizzare una GUI (Graphic User Interface) completa con la quale sarà possibile controllare Arduino.
Il sito internet di riferimento è *http://processing.org*, da cui è possibile scaricare Processing e trovare tutte le informazioni del caso, esempi, riferimenti, librerie, libri consigliati; rimandiamo ad esso per qualsiasi approfondimento relativamente agli argomenti trattati in questo corso.

SCARICARE ED INSTALLARE PROCESSING

Si comincia visitando il sito *http://processing.org/download* e selezionando la versione relativa al sistema operativo che si sta utilizzando: Windows, Mac, Linux. Processing è indipendente dal sistema operativo e il codice creato con esso può essere trasportato da un sistema all'altro senza alcuna modifica.
L'ultima versione di Processing disponibile nel momento in cui scriviamo questo corso è la Processing-1.2.1; in ogni caso, nel sito sono sempre disponibili le versioni precedenti.
Per quanto riguarda la versione Windows, ci sono due modalità:
1) Windows;
2) Windows (Without Java).

La prima modalità è quella standard, mentre la seconda presuppone l'installazione da soli di JDK (Java Development Kit), che è diverso dal comune JRE (Java Run Time Engine). Sostanzialmente, Processing si fonda su Java ed in un certo senso ne è una derivazione, pertanto per funzionare correttamente necessita di alcune componenti, che nella modalità standard vengono installate automaticamente, mentre nella seconda modalità prevista bisogna farlo da soli.
La seconda modalità di installazione deve essere seguita soltanto se si ha dimestichezza con Java e con tutto il Development Kit (ovviamente se per altri scopi abbiamo già installato JDK, seguiremo questa modalità).
Una volta scaricato, Processing si presente-

rà come un file .zip, quindi occorre estrarlo e posizionare la cartella in una locazione dell'HD; a questo punto basta fare do ppio clic su *processing.exe* per cominciare.

PROCESSING DEVELOPMENT ENVIRONMENT (PDE)

Una volta avviato il Processing Development Environment, si presenterà un'area chiamata Text Editor, dove verranno scritti i programmi; appena sopra quest'area c'è una fila di pulsanti denominata Toolbar. Sotto al Text Editor c'è la Message Area e sotto a questa la Console. La Message Area è usata per inviare da Processing i messaggi di errore, mentre la Console serve per trasmettere, ad esempio, dei messaggi dal programma in esecuzione. La Fig. 1 mostra il PDE.
I programmi creati con Processing vengono chiamati sketch, termine familiare a chi ha seguìto il nostro corso su Arduino.
La Toolbar è composta dai seguenti pulsanti:

1. RUN – avvia il programma;
2. STOP – ferma il programma;
3. NEW – apre un altro editor;
4. OPEN – apre un altro programma;
5. SAVE – salva il lavoro corrente; il formato di un file in Processing è .pde;
6. EXPORT – crea una cartella unica con tutto il lavoro realizzato, che può essere installata su un web-server ed è composta da un file .pde che è il codice sorgente; il file .jar sarà il programma, mentre il file .html corrisponde alla pagina web.

In EXPORT, facendo doppio clic sul file *index.html* si avvierà il browser web, mostrando lo sketch creato.
Nel menù *File* si trova anche il comando Export to Application, che crea un'applicazione per il sistema operativo che vogliamo: Mac, Windows, Linux. L'applicazione creata sarà quindi trasportabile ed installabile su altri PC che non hanno Processing.
Scrivere codice coinvolge spesso lo studio e la vera e propria esplorazione di esempi già fatti; sotto questo aspetto, Processing già di suo ha tantissimi esempi molto ben strutturati in argomenti e categorie: basta impartire il comando di menu File –> Examples ed eseguire i vari codici sorgente disponibili.
Un altro strumento indispensabile per iniziare a programmare con Processing è il Processing Reference, cioè la lista di tutte le primitive del linguaggio con tanto di spiegazione e breve codice di esempio. Per accedere al Refence basta andare sul menu Help ed impartire il comando Reference, allorché si aprirà la pagina html con la lista di tutte le API del linguaggio Processing; basta fare clic su una di esse e si aprirà la relativa pagina con la descrizione ed il codice di esempio.

CODICE

I programmi di esempio che proporremo in questo corso sono i seguenti:

1. LEZIONE 1_1 FORME BASE; questo programma mostra come disegnare linee, rettangoli, cerchi, quadrilateri sull'area di disegno;
2. LEZIONE 1_2 FORME BASE 2; rispetto al programma precedente, aggiunge le funzionalità legate al tratto di disegno ed al riempimento di colore delle figure;
3. LEZIONE 1_3 COLORI BASE; illustra le funzionalità legate a colori, riempimenti e trasparenze;
4. LEZIONE 1_4 ELETTRINO 1; questo programma permette di disegnare un robot con le funzionalità descritte nei programmi precedenti;
5. LEZIONE 1_5 STRUTTURA BASE DI UN PROGRAMMA; in esso viene mostrata la struttura base di un programma scritto in Processing;
6. LEZIONE 1_6 CONTROLLO DI FLUSSO_RIPETIZIONI; questo programma descrive l'uso dei cicli FOR;
7. LEZIONE 1_7 CONTROLLO DI FLUSSO_IF; questo programma mostra l'utilizzo del costrutto condizionale IF;
8. LEZIONE 1_8 USO DEL TESTO; aggiungiamo del testo al programma dell'esercizio precedente;
9. LEZIONE 1_9 - ANIMAZIONE; realizziamo una semplice animazione.

Partiamo dunque dal primo programma di esempio.

LEZIONE 1_1 FORME BASE
Iniziamo ora a scrivere il nostro primo programma, che consiste semplicemente

Listato 1

```
size(260,400);                        //dimensioni X,Y dell'area di disegno
background(205);                      //sfondo dell'area di disegno, in gradazione di grigio
smooth();                             //forme disegnate meno spigolose
strokeWeight(2);                      //spessore del tratto di disegno pin px

line(60,60,200,20);                   //linea X1,Y1, X2,Y2
triangle(200,60,200,100,60,100);      //triangolo X1,Y1,X2,Y2,X3,Y3
quad(100,120,180,140,160,160,60,160); //quadrilatero X1,Y1,X2,Y2,X3,Y3,X4,Y4
rect(80,200,100,40);                  //rettangolo X,Y, larghezza, altezza
ellipse(120,280,50,50);               //ellisse X,Y, larghezza, altezza
arc(120, 350, 70, 70, 0, PI+HALF_PI); //arco X,Y, largh, Altezza, ang inizio, ang fine
```

nel rappresentare sullo schermo delle figure geometriche utilizzando le funzioni native di Processing.
La dimensione dell'area in cui andremo a disegnare le nostre forme è definita dalla funzione: size(), le dimensioni dell'area di disegno sono definite in pixel e i parametri da passare alla funzione sono *width* ed *eight*, ovvero larghezza ed altezza.
L'origine 0,0 del piano di disegno è posta nello spigolo in alto a sinistra, la direzione di incremento dell'asse X è verso destra, mentre la direzione di incremento dell'asse Y è verso il basso.
La funzione background() permette di definire il colore di sfondo del piano di disegno; inserendo un solo valore che va da 0 a 255, Processing riconoscerà che stiamo lavorando in scala di grigi. Successivamente affronteremo il tema dei colori.
Il comando smooth() evita la spigolosità delle varie forme disegnate, mentre strokeWeight() definisce lo spessore del tratto in pixel. Le linee vengono disegnate con il comando line(), che si aspetta le coordinate del punto di partenza (X1,Y1) e di quello di arrivo (X2,Y2). In maniera del tutto similare triangle() disegna dei triangoli e dobbiamo definire le coordinate dei tre vertici (X1,Y1,X2,Y2,X3,Y3). Per disegnare dei quadrilateri si usa il comando quad(), inserendo le coordinate dei quattro vertici. Diverso è il caso per i rettangoli, che sono disegnati con il comando rect(), i cui parametri sono le coordinate X,Y dello spigolo in alto a sinistra, la larghezza del rettangolo e la sua altezza. In maniera del tutto simile a come vengono disegnati i rettangoli, si tracciano

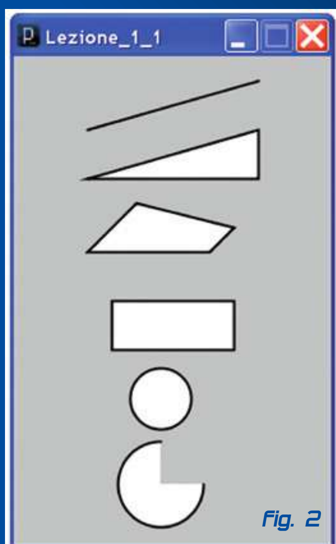


Fig. 2

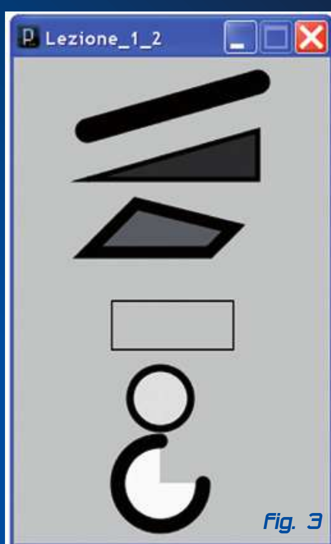


Fig. 3

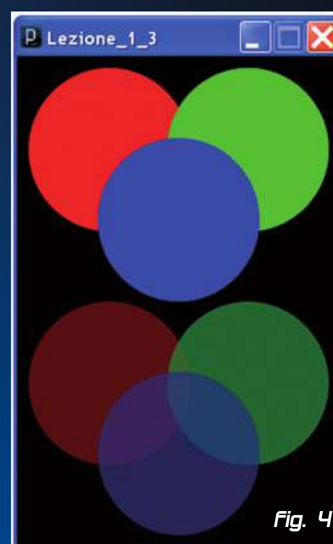


Fig. 4

le ellissi; il comando corrispondente è ellipse(), i cui parametri sono le coordinate X,Y del centro la larghezza e l'altezza (ovviamente se altezza e larghezza sono identiche otterremo un cerchio).
L'ultimo comando che esaminiamo è arc(), il cui funzionamento è identico a ellipse(), ma ha due parametri in più, che sono l'angolo di inizio e l'angolo di fine (espressi in radianti e con verso positivo di crescita quello antiorario).
Il codice corrispondente è quello visibile nel Listato 1.

LEZIONE 1_2 FORME BASE 2
Il codice di questo secondo programma aggiunge due particolari funzionalità a quello precedente, o meglio, prima di ogni primitiva di disegno definisce fill() e strokeWeight(). Ne consegue che ogni forma disegnata avrà il suo colore di riempimento

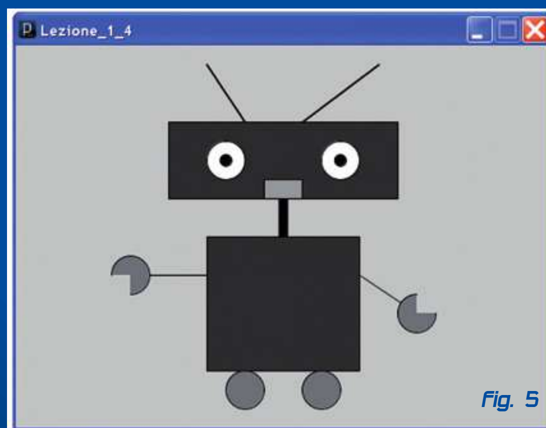


Fig. 5

ed il suo spessore del tratto di disegno. La Fig. 2 mostra le forme base del programma precedente, mentre la Fig. 3 illustra le forme base con le nuove funzionalità appena descritte.

LEZIONE 1_3 COLORI BASE
Il codice di questo terzo programma mostra come utilizzare i colori e le funzionalità di Processing relativamente alle trasparenze; con esso vengono disegnate due combina-

Listato 2

```
size(280,420);  //dimensioni X,Y dell'area di disegno
                //l'origine è situata nello spigolo in alto a sinistra,
                //X incrementa da sinistra a destra          0 ---> X
                //Y incrementa dall'alto verso il basso      0 |
                //                                             |
                //                                             V
                //                                           Y


//i colori sono R - G - B
//i valori che possono assumere vanno da 0 --> 255
//nella funzione fill() possiamo mettere un solo parametro, quindi stiamo lavorando in gradazioni di grigio
//nella funzione fill() possiamo mettere tre parametri, quindi lavoreremo in RGB

background(0,0,0);//sfondo dell'area di disegno nero R=0, G=0, B=0

smooth(); //questa funzione permette di rendere le forme disegnate meno spigolose

noStroke(); //con questa funzione le forme disegnate sono prive di contorno

fill(255, 0, 0);          //riempimento di colore rosso
ellipse(80,80,140,140);   //prima circonferenza
fill(0, 255, 0);          //riempimento di colore verde
ellipse(200,80,140,140);  //seconda circonferenza
fill(0, 0, 255);          //riempimento di colore blu
ellipse(140,140,140,140); //terza circonferenza


//oltre ai colori si può impostare la trasparenza
//i valori vanno da 0 --> 255
//nella funzione fill() possiamo mettere quattro parametri: R,G,B, trasparenza

fill(255, 0, 0,100);       //riempimento di colore rosso e trasparenza 100
ellipse(80,280,140,140);   //prima circonferenza
fill(0, 255, 0, 100);      //riempimento di colore verde e trasparenza 100
ellipse(200,280,140,140);  //seconda circonferenza
fill(0, 0, 255, 100);      //riempimento di colore blu e trasparenza 100
ellipse(140,340,140,140);  //terza circonferenza
```

zioni di tre cerchi con i colori base. Nella prima sequenza i colori sono pieni e non ci sono trasparenze, mentre nella seconda vengono utilizzate le trasparenze Fig. 4. Trovate il codice corrispondente a questo esempio nel Listato 2.

LEZIONE 1_4 ELETTRINO 1

A questo punto mettiamo assieme tutte le funzioni descritte ed incominciamo a realizzare un disegno.

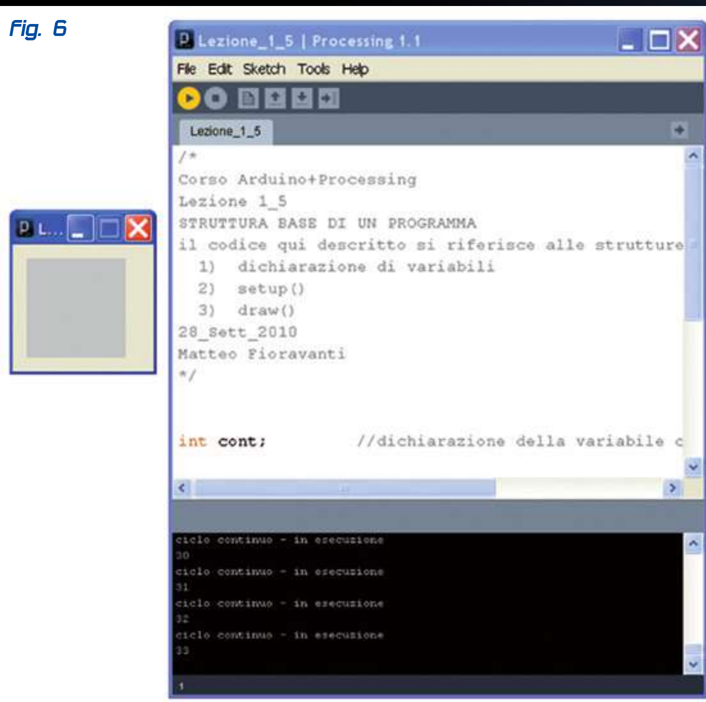
Il codice di questo quarto programma permette di combinare le forme base per disegnare un piccolo robot (Fig. 5) cui diamo il nome di Elettrino e che richiameremo più avanti in successivi programmi, nei quali saranno aggiunte altre funzionalità.

LEZIONE 1_5 STRUTTURA BASE DI UN PROGRAMMA

Il quinto programma esegue un ciclo in cui nel terminale viene scritto continuamente un numero che viene incrementato ad ogni ciclo (Fig. 6). La struttura di un programma scritto in Processing si compone di tre parti fondamentali:

- la dichiarazione delle variabili;
- void setup(){} che è una parte di codice eseguito una sola volta;

Fig. 6



Listato 3

```
int cont;         //dichiarazione della variabile cont come intero

//setup viene eseguito una sola volta
void setup(){
  cont=0;         //inizializzazione della variabile cont
  println("SETUP - eseguito una sola volta");  //scrittura a terminale
}

//draw è un ciclo che viene eseguito continuamente
void draw(){
  println("ciclo continuo - in esecuzione");  //scrittura a terminale
  println(cont);  //scrittura a terminale del valore di cont
  delay(1000);    //attesa di 1 secondo, il valore di delay() è espresso in ms
  cont +=1;          //incremento di cont di 1 ad ogni ciclo
}
```

Listato 4

```
size(480,120);
smooth();
background(0,0,0);      // sfondo nero
strokeWeight(1);

//ciclo for
for (int i = 40; i<480; i +=40){ //variabile di conteggio i, che parte da 40 fino a 280, con incrementi di 40 per ogni ciclo
  fill(50+i/2, 90+i/10, 200);    //riempimento, i colori variano in modo proporzionale con l'incremento di i
  ellipse (i, 60, 60, 60);       //vengono disegnati cerchi, la cui posizione X è i
  }
```

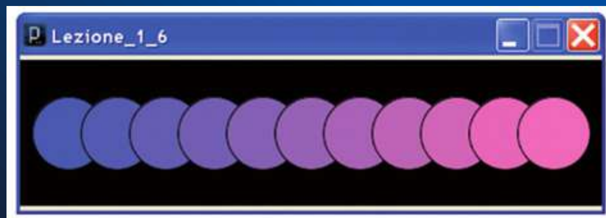


Fig. 7

- void draw(){} che invece è la parte di codice eseguita continuamente.

Nel programma appena descritto (Listato 3) è stata utilizzata la funzione delay(1000); che permette di creare un ritardo di 1 secondo; in pratica l'argomento della funzione è un numero che esprime il ritardo in ms.
La scrittura cont +=1; significa che la variabile cont viene incrementata di un'unità ad ogni ciclo.

LEZIONE 1_6 CONTROLLO DI FLUSSO_ RIPETIZIONI
Questo sesto programma introduce l'utilizzo dei controlli di flusso. Il codice (Listato 4) serve a disegnare una ripetizione di cerchi di colore diverso, come mostrato nella Fig. 7; questa funzionalità è ottenuta attraverso un ciclo FOR. Quest'ultimo è descritto dalla variabile che viene incrementata ad ogni ripetizione, dalla condizione per cui il ciclo viene ripetuto (in pratica, finché la condizione è vera il ciclo viene ripetuto, mentre quando diventa falsa il programma esce dal ciclo) ed infine dall'incremento della variabile.
Nel caso del programma in questione, la variabile che viene incrementata è i, definita come intero for(int i ….
La condizione per cui il ciclo è ripetuto è: "fintanto che i è minore di 480" for (int i; i< 480….
Invece l'incremento della variabile i ad ogni ciclo è 40: for (int i; i< 480; i +=40).

LEZIONE 1_7 CONTROLLO DI FLUSSO_IF
Il settimo programma della prima lezione permette di disegnare un quadrato con all'interno quattro quadranti; spostando il cursore del mouse sopra un quadrante, questo cambia di colore (Fig. 8).
Lo scopo principale di questo programma è verificare la posizione del cursore del mouse; se questa è entro i limiti definiti di un quadrante viene cambiato il colore



Fig. 8



Fig. 9

dell'area. Definite le dimensioni dell'area di lavoro (con size(200,200)), la posizione X ed Y del mouse può essere ottenuta semplicemente con i comandi mouseX e mouseY, i quali ci restituiranno, nell'istante in cui sono richiamati, il valore X ed Y del cursore del mouse nell'area di disegno.
La condizione di verifica del primo quadrante è la seguente:

```
if ((mouseX > x1) && (mouseX < x2) && (mouseY > y1) && (mouseY < y3)){..
```

In pratica si tratta di una condizione con quattro AND logici (&&) che verifica se la posizione X del mouse è tra X1 e X2 e se la posizione Y del mouse è tra Y1 e Y2; se tutte e quattro le condizioni sono verificate, la condizione complessiva è vera e di conseguenza viene colorata l'area (Listato 5).

LEZIONE 1_8 USO DEL TESTO
Questo programma aggiunge, rispetto al precedente, del testo per identificare i quadranti (Fig. 9); il comando qui utilizzato è: text("testo da scrivere", posizione X,

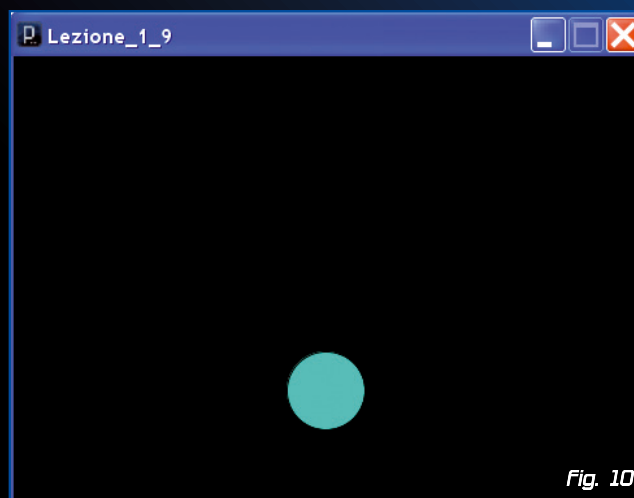


Fig. 10

Listato 5

```
int x1 = 0;
int y1 = 0;

int x2 = 100;
int y2 = 0;

int x3 = 100;
int y3 = 100;

int x4 = 0;
int y4 =100;

int h = 100;
int l = 100;

void setup()
{
  size(200,200);
}

void draw()
{
//costruzione rettangolo QUADRANTE 1
  if ((mouseX > x1) && (mouseX < x2) && (mouseY > y1) _
  && (mouseY < y3))
  { //condizione di verifica se il mouse è nel QUADRANTE 1
     fill(0);
   } else
     { fill(255); }
   rect(x1,y1,l,h);

//costruzione rettangolo QUADRANTE 2
  if ((mouseX > x2) && (mouseX < width) && _
  (mouseY > y2) && (mouseY < y4))
  { //condizione di verifica se il mouse è nel QUADRANTE 2
     fill(0);
   } else
      { fill(255); }
    rect(x2,y1,l,h);

//costruzione rettangolo QUADRANTE 3
  if ((mouseX > x2) && (mouseX < width) && _
  (mouseY > y3) && (mouseY < height))
  { //condizione di verifica se il mouse è nel QUADRANTE 3
    fill(0);
   } else
     { fill(255); }
   rect(x3,y3,l,h);

//costruzione rettangolo QUADRANTE 4
  if ((mouseX > x4) && (mouseX < x3) && _
  (mouseY > y4) && (mouseY < height))
  { //condizione di verifica se il mouse è nel QUADRANTE 4
    fill(0);
   } else
     { fill(255); }
   rect(x4,y4,l,h);
}
```

posizioneY). Il Listato 6 mostra l'utilizzo di tale comando nell'applicazione oggetto di questo esercizio.

Listato 6

```
...
text("x1,y1",15+x1,15+y1);
text("x2,y2",15+x2,15+y2);
text("x3,y3",15+x3,15+y3);
text("x4,y4",15+x4,15+y4);
...
```


Fig. 11

LEZIONE 1_9 - ANIMAZIONE

Realizziamo ora una semplice animazione consistente in un'ellisse che oscilla in verticale tra due posizioni, come mostra la Fig. 10. Il programma si basa sul concetto fondamentale in Processing, secondo cui void draw(){} viene ripetuto continuamente; per la precisione, tale funzione è richiamata, di regola, 60 volte al secondo. Pertanto sfruttando questo principio possiamo realizzare delle animazioni.
La tecnica base è quella di creare una figura in una certa posizione, quindi cancellarla nel frame successivo e ridisegnarla in una posizione differente dalla prima.
L'oscillazione sull'asse verticale della posizione dell'ellisse è ottenuta mediante la formula:

```
float y1 = offset + (sin(angolo))*scala;
```

La posizione è definita da y1, angolo è una variabile che viene incrementata ad ogni ciclo, mentre la funzione sin() (funzione seno di un angolo espresso in radianti) restituisce un numero compreso tra -1 e 1. La variabile scala moltiplica sostanzialmente il valore della funzione sin(), mentre offset rappresenta la posizione iniziale attorno alla quale avviene l'oscillazione.
La variabile di angolo verrà incrementata indefinitamente (perlomeno fino alla sua totale occupazione del campo di memoria e poi ricomincerà), ma la funzione seno è periodica di 2pi, quindi il numero che ne consegue dal continuo incremento di angolo sarà un multiplo di 2pi ed il risultato della funzione seno sarà sempre compreso tra -1 e 1.
Variando le costanti iniziali, ossia angolo,

Listato 7

```
float angolo = 0.0;  //angolo
float offset = 200;  //posizione iniziale dell'ellisse
float scala = 60;    //ampiezza dell'oscillazione
float velocita = 0.02;  //velocità dell'oscillazione


void setup(){
  size(560,400);
  smooth();
  strokeWeight(1);
  background(0);
  ellipseMode(RADIUS);
}

void draw(){

  fill(0,0,0);          //riempimento di colore nero
  rect(0,0,560,400);    //disegno di un rettangolo di colore nero di dimensioni pari
                                  all'area di disegno

  float y1 = offset + (sin(angolo))*scala;    //calcolo della nuova posizione Y dell'ellisse
  fill(0,220,200);                //  colore di riempimento dell'ellisse

  ellipse(280, y1, 30, 30);  //disegno dell'ellisse

  angolo += velocita;   //incremento ad ogni ciclo della variabile angolo

}
```

offset, scala e velocità, si può sperimentare cosa accade. Il Listato 7 mostra il codice corrispondente a questo esempio.
Bene, con gli esercizi abbiamo terminato. In questa prima puntata del corso su Processing abbiamo brevemente descritto come installare questo linguaggio di programmazione e dei brevi programmi di esempio; nella prossima incominceremo a realizzare delle applicazioni con Arduino. Scriveremo quindi dei semplici programmi che tratteranno l'interfacciamento di un software scritto in Processing con un piccolo sistema basato su Arduino. Fondamentalmente, realizzeremo su Processing un'interfaccia grafica e faremo accendere un LED su Arduino; l'applicazione sarà molto semplice, ma alla base ci sarà la costruzione di una struttura master/slave e di un protocollo di comunicazione tra i due sistemi.

EXTRA – PLASMA FADING

La finalità di Processing è, come già detto, costituire un linguaggio semplice per applicazioni di grafica e design; il codice *plasma_fadig.pde* allegato ai programmi di esempio di questo corso si riferisce proprio a ciò. Si tratta di un'applicazione in cui viene disegnato un raggio colorato che ruota a tutto schermo (Fig. 11).
Siete invitati a variare i parametri del codice per sperimentare cosa accade in pratica.